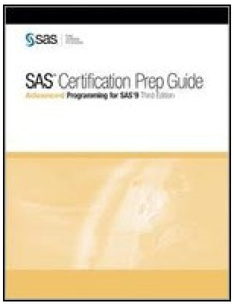


SAS Certification Prep Guide: Advanced Programming for SAS 9, Third Edition

by SAS Institute

Chapter 16: Using Lookup Tables to Match Data

Overview

Introduction

Sometimes, you need to combine data from two or more sets into a single observation in a new data set according to the values of a common variable. When the data sources are two or more data sets that have a common structure, you can use a match-merge to combine the data sets. However, in some cases the data sources do not share a common structure. When data sources do not have a common structure, you can use a lookup table to match them. A *lookup table* is a table that contains key values.

Key	Var_X
1998	X1
1999	X2
2000	X3
2001	X4
2002	X5

+

Key	Var_Y	Var_Z
1998	Y1	Z1
1999	Y2	Z2
2000	Y3	Z3
2001	Y4	Z4
2002	Y5	Z5

=

Year	Var_X	Var_Y	Var_Z
1998	X1	Y1	Z1
1999	X2	Y2	Z2
2000	X3	Y3	Z3
2001	X4	Y4	Z4
2002	X5	Y5	Z5

The technique that you use to perform a table lookup is dependent on your data. This chapter focuses on using multidimensional arrays to perform table lookups and transposing SAS data sets in preparation for a match-merge.

Objectives

In this chapter, you learn to

- use a multidimensional array to match data
- use stored array values to match data
- use PROC TRANSPOSE to transpose a SAS data set and prepare it for a table lookup
- merge a transposed SAS data set
- use a hash object as a lookup table (for SAS *9* and later).

Using Multidimensional Arrays

Review of the Multidimensional Array Statement

When a lookup operation depends on more than one numerical factor, you can use a multidimensional array. You use an ARRAY statement to create an array. The ARRAY statement defines a set of elements that you plan to process as a group.

General form, multidimensional ARRAY statement:

```
ARRAY array-name {rows,cols,…} <$><length>
        <array-elements> <(initial values)>;
```

where

array-name

names the array.

rows

specifies the number of array elements in a row arrangement.

cols

specifies the number of array elements in a column arrangement.

array-elements

names the variables that make up the array.

initial values

specifies initial values for the corresponding elements in the array that are separated by commas or spaces.

Note The keyword *_TEMPORARY_* might be used instead of *array-elements* to avoid creating new data set variables. Only temporary array elements are produced as a result of using _TEMPORARY_.

When you are working with arrays, remember that

- the name of the array must be a SAS name that is not the name of a SAS variable in the same DATA step
- the variables listed as array elements must all be the same type (either all numeric or all character)
- the initial values specified can be numbers or character strings. You must enclose all character strings in quotation marks.

Note If you use the _TEMPORARY_ keyword in an array statement, remember that temporary data elements behave like DATA step variables with the following exceptions:

- They do not have names. Refer to temporary data elements by the array name and dimension.
- They do not appear in the output data set.
- You cannot use the special subscript asterisk (*) to refer to all the elements.
- Temporary data element values are always automatically retained, rather than being reset to missing at the beginning of the next iteration of the DATA step.

Example

Suppose you need to determine the wind chill values for the flights represented in the SAS data set *Sasuser.Flights*. The data set contains three variables: **`Flight`** (the flight number), **`Temp`** (the average outdoor temperature during the flight), and **`Wspeed`** (the average wind speed during the flight).

Obs	Flight	Temp	Wspeed
1	IA2736	-8	9
2	IA6352	-4	16

Figure 16.1: SAS Data Set Sasuser.Flights

Wind chill values are derived from the air temperature and wind speed as shown in the following wind chill lookup table. To determine the wind chill for each flight, you can create a multidimensional array that stores the wind chill values shown in the table. You can then match the values of **`Temp`** and **`Wspeed`** with the wind chill values stored in the array.

Wind Speed (in miles per hour)

	-10	-5	0	5	10	15	20	25	30
5	-22	-16	-11	-5	1	7	13	19	25
10	-28	-22	-16	-10	-4	3	9	15	21
15	-32	-26	-19	-13	-7	0	6	13	19
20	-35	-29	-22	-15	-9	-2	4	11	17
25	-37	-31	-24	-17	-11	-4	3	9	16
30	-39	-33	-26	-19	-12	-5	1	8	15
35	-41	-34	-27	-21	-14	-7	0	7	14
40	-43	-36	-29	-22	-15	-8	-1	6	13

Figure 16.2: Temperature (in degrees Fahrenheit)

In the following program, the ARRAY statement creates the two-dimensional array **WC** and specifies the dimensions of the array: four rows and two columns. No variables are created from the array because the keyword *_TEMPORARY_* is used. The initial values specified correspond to the values in the wind chill lookup table. For this example, only the values in the first two columns and four rows in the wind chill lookup table are included in the array.

```
data work.wndchill (drop = column row);
  array WC{4,2} _temporary_
        (-22,-16,-28,-22,-32,-26,-35,-29);
  set sasuser.flights;
  row = round(wspeed,5)/5;
  column = (round(temp,5)/5)+3;
  WindChill= wc{row,column};
run;
```

Wind Speed (in miles per hour)

	-10	-5	0	5	10	15	20	25	30
5	-22	-16	-11	-5	1	7	13	19	25
10	-28	-22	-16	-10	-4	3	9	15	21
15	-32	-26	-19	-13	-7	0	6	13	19
20	-35	-29	-22	-15	-9	-2	4	11	17
25	-37	-31	-24	-17	-11	-4	3	9	16
30	-39	-33	-26	-19	-12	-5	1	8	15
35	-41	-34	-27	-21	-14	-7	0	7	14
40	-43	-36	-29	-22	-15	-8	-1	6	13

Figure 16.3: Temperature (in degrees Fahrenheit)

The value of **WindChill** for each flight is determined by referencing the array based on the values of **Wspeed** and **Temp** in the *Sasuser.Flights* data set. The row number for the array reference is determined by the value of **Wspeed.** The column number for the array reference is determined by the value of **Temp.**

Table Representation of the WC Array

```
data work.wndchill (drop = column row);
  array WC{4,2} _temporary_
```

```
          (-22,-16,-28,-22,-32,-26,-35,-29)
    set sasuser.flights;
  row = round(wspeed,5)/5;
  column = (round(temp,5)/5)+3;
  WindChill= wc{row,column};
  run;
```

-22	-16
-28	-22
-32	-26
-35	-29

The rounding unit for the value of **Wspeed** is 5 because the values for wind speed in the wind chill table are rounded to every 5 miles-per-hour. **Wspeed** is then divided by 5 to derive the row number for the array reference.

Like the value for **Wspeed**, the value of **Temp** is rounded to the nearest 5, then divided by 5. The offset of 3 is added to the value because the third column in the wind chill lookup table represents 0 degrees.

```
data work.wndchill (drop = column row);
  array WC{4,2} _temporary_
        (-22,-16,-28,-22,-32,-26,-35,-2 9);
  set sasuser.flights;
  row = round(wspeed,5)/5;
  column = (round(temp,5)/5)+3;
  WindChill= wc{row,column};
run;
```

PROC PRINT output shows the completed data set.

```
proc print data=work.wndchill;
run;
```

Obs	flight	temp	wspeed	WindChill
1	IA2736	-8	9	-28
2	IA6352	-4	16	-26

Using Stored Array Values

Overview

In the previous section, the wind chill values were loaded into the **WC** array when the array was created. In some cases, you might need to store array values in a SAS data set rather than loading them in an ARRAY statement. Array values should be stored in a SAS data set when

- there are too many values to initialize easily in the array
- the values change frequently
- the same values are used in many programs.

Example

Suppose you want to compare the *actual* cargo revenue values in the SAS data set *Sasuser.Monthsum* to the *target* cargo revenue values in the SAS data set *Sasuser.Ctargets*.

Sasuser.Monthsum contains the actual cargo and passenger revenue figures for each month from 1997 through 1999.

Table 16.1: SAS Data Set Sasuser.Monthsum (first five observations of selected variables)

Obs	SaleMon	RevCargo	MonthNo
1	JAN1997	$171,520,869.10	1
2	JAN1998	$238,786,807.60	1
3	JAN1999	$280,350,393.00	1
4	FEB 1997	$177,671,530.40	2
5	FEB1998	$215,959,695.50	2

The SAS data set *Sasuser.Ctargets* contains the target cargo revenue figures for each month from 1997 through 1999.

Table 16.2: SAS Data Set Sasuser.Ctargets

Obs	Year	Jan	Feb	Mar	Apr	May	Jun
1	1997	192284420	86376721	28526103	260386468	109975326	102833104
2	1998	108645734	147656369	202158055	41160707	264294440	267135485
3	1999	85730444	74168740	39955768	312654811	318149340	187270927

Obs	Jul	Aug	Sep	Oct	Nov	Dec
1	196728648	236996122	112413744	125401565	72551855	136042505
2	208694865	83456868	286846554	275721406	230488351	24901752
3	123394421	34273985	151565752	141528519	178043261	181668256

You want to create a new SAS data set, *Work.Lookupl*, that lists the actual and target values for each month. *Work.Lookupl* should have the same structure as *Sasuser.Monthsum*: an observation for each month and year, as well as a new variable, `Ctarget` (target cargo revenues). The value of `Ctarget` is derived from the target values in *Sasuser.Ctargets*.

Table 16.3: SAS Data Set Work.Lookupl (first five observations of selected variables)

Obs	SaleMon	RevCargo	Ctarget
1	JAN1997	$171,520,869.10	$192,284,420.00
2	JAN1998	$238,786,807.60	$108,645,734.00
3	JAN1999	$280,350,393.00	$85,730,444.00
4	FEB 1997	$177,671,530.40	$86,376,721.00
5	FEB1998	$215,959,695.50	$147,656,369.00

Sasuser.Monthsum and *Sasuser.Ctargets* cannot be merged because they have different structures:

- *Sasuser.Monthsum* has an observation for each month and year.
- *Sasuser.Ctargets* has one column for each month and one observation for each year.

However, the data sets have two common factors: month and year. You can use a multidimensional array to match the actual values for each month and year in *Sasuser.Monthsum* with the target values for each month and year in *Sasuser.Ctargets*.

Creating an Array

The first step is to create an array to hold the values in the target data set, *Sasuser.Ctargets*. The array needs two dimensions: one for the year values and one for the month values. In the following program, the first ARRAY statement creates the two-dimensional array, `Targets.`

Remember that the index of an array does not have to range from one to the number of elements. You can specify a range for the values for the index when you define the array. In this case, the dimensions of the array are specified as three rows

(one for each year: 1997, 1998, and 1999) and 12 columns (one for each month).

```
data work.lookup1;
  array Targets{1997:1999,12}_temporary_;
  if _n_=1 then do i= 1 to 3;
             set sasuser.ctargets;
             array mnth{*} Jan--Dec;
             do j=1 to dim(mnth);
                targets{year,j}=mnth{j};
             end;
  end;
  set sasuser.monthsum(keep=salemon revcargo monthno);
  year=input(substr(salemon,4),4.);
  Ctarget=targets{year,monthno};
  format ctarget dollar15.2;
run;
```

The following table represents the **Targets** array. Notice that the array is not populated. The next step is to load the array elements from *Sasuser.Ctargets*

Table 16.4: Table Representation of Targets Array

	1	2	3	4	5	6	7	8	9	10	11	12
1997												
1998												
1999												

Note The row dimension for the **Targets** array could have been specified using the value *3*. For example,

```
array Targets{3,12} _temporary_;
```

However, using the notation **1997:1999** simplifies the program by eliminating the need to map numeric values to the year values.

Loading the Array Elements

The **Targets** array needs to be loaded with the values in *Sasuser.Ctargets*. One method for accomplishing this task is to load the array within a DO loop.

Table 16.5: SAS Data Set Sasuser.Ctargets

Year	Jan	Feb	Mar	Apr	May	Jun
1997	192284420	86376721	28526103	260386468	109975326	102833104
1998	108645734	147656369	202158055	41160707	264294440	267135485
1999	85730444	74168740	39955768	312654811	318149340	187270927

	Jul	Aug	Sep	Oct	Nov	Dec
	196728648	236996122	112413744	125401565	72551855	136042505
	208694865	83456868	286846554	275721406	230488351	24901752
	123394421	34273985	151565752	141528519	178043261	181668256

The IF-THEN statement specifies that the **Targets** array is loaded only once, during the first iteration of the DATA step. The DO loop executes three times, once for each observation in *Sasuser.Ctargets*.

The ARRAY statement within the DO loop creates the **Mn th** array, which will be used to store the elements from *Sasuser.Ctargets*. The dimensions of the **Mnth** array are specified using an asterisk, which enables SAS to automatically count the array elements.

Note If you use an asterisk to specify the dimensions of an array, you must list the array elements. You cannot use an asterisk to specify an array's dimensions if the elements of the array are specified with the _TEMPORARY_ keyword.

The array elements **Jan** through **Dec** are listed using a double hyphen (- -). The double hyphen (- -) is used to read the specified values based on their positions in the PDV instead of alphabetically.

```
data work.lookup1;
   array Targets{1997:1999,12} _temporary_;
   if _n_=1 then do i= 1 to 3;
              set sasuser.ctargets;
              array Mnth{*} Jan--Dec;
              do j=1 to dim(mnth);
                 targets{year,j}=mnth{j};
              end;
   end;
   set sasuser.monthsum(keep=salemon revcargo monthno);
   year=input(substr(salemon,4),4.);
   Ctarget=targets{year,monthno};
   format ctarget dollar15.2;
run;
```

The following table shows the values in the **Mnth** array after the first iteration of the DO loop.

Table 16.6: Table Representation of Mnth Array (after the first iteration of the DO loop)

Jan	Feb	Mar…	…Oct	Nov	Dec
192284420	86376721	260386468	125401565	72551855	136042505

Within the nested DO loop, the **Targets** array reference is matched to the **Mnth** array reference in order to populate the **Targets** array. The DIM function returns the number of elements in the **Mnth** array (in this case *12)* and provides an ending point for the nested DO loop.

```
data work.lookup1;
  array Targets{1997:1999,12} _temporary_;
  if _n_=1 then do i= 1 to 3;
             set sasuser.ctargets;
             array Mnth{*} Jan--Dec;
             do j=1 to dim(mnth);
                targets{year,j}=mnth{j};
             end;
  end;
  set sasuser.monthsum(keep=salemon revcargo monthno);
  year=input(substr(salemon,4),4.);
  Ctarget=targets{year,monthno};
  format ctarget dollar15.2;
run;
```

Table 16.7: Table Representation of Mnth Array (after the second iteration of the DO loop)

Jan	Feb	Mar…	…Oct	Nov	Dec
108645734	147656369	202158055	275721406	230488351	24901752

Table 16.8: Table Representation of Mnth Array (after the third iteration of the DO loop)

Jan	Feb	Mar…	…Oct	Nov	Dec
85730444	74168740	39955768	141528519	178043261	181668256

Table 16.9: Table Representation of Populated Targets Array

	1	2	3…	…10	11	12
1997	192284420	86376721	260386468	125401565	72551855	136042505
1998	108645734	147656369	202158055	275721406	230488351	24901752
1999	85730444	74168740	39955768	141528519	178043261	181668256

Note The dimension of the **Mnth** array could also be specified using the numeric value *12*. However, the asterisk

notation enables the program to be more flexible. For example, using the asterisk, the program would not need to be edited if the target data set contained data for only eleven months. Remember that if you use an asterisk to count the array elements, you must list the array elements.

Reading the Actual Values

The last step is to read the actual values stored in *Sasuser.Monthsum*. Remember that you need to know the month and year values for each observation in order to locate the correct target revenue values.

Table 16.10: SAS Data Set Sasuser.Monthsum (first five observations of selected variables)

SaleMon	RevCargo	MonthNo
JAN1997	$171,520,869.10	1
JAN1998	$238,786,807.60	1
JAN1999	$280,350,393.00	1
FEB1997	$177,671,530.40	2
FEB1998	$215,959,695.50	2

The values for month are read in from **`MonthNo.`** The year values are contained within the values of **`SaleMon`** and can be extracted using the SUBSTR function. In this example, the SUBSTR function brings in four characters from **`SaleMon`**, starting at the fourth character. Note that the INPUT function is used to convert the value that is extracted from **`SaleMon`** from character to numeric in the assignment statement for **`Year.`** A numeric format must be used because the value of **`Year`** will be used as an array reference.

The values of **`Ctarget`** are then read in from the **`Targets`** array based on the value of **`Year`** and **`MonthNo.`**

```
data work.lookup1;
  array Targets{1997:1999,12} _temporary_;
  if _n_=1 then do i= 1 to 3;
              set sasuser.ctargets;
              array Mnth{*} Jan--Dec;
              do j=1 to dim(mnth);
                 targets{year,j}=mnth{j};
              end;
  end;
  set sasuser.monthsum(keep=salemon revcargo monthno);
  year=input(substr(salemon,4),4.);
  Ctarget=targets{year,monthno};
  format ctarget dollar15.2;
run;
```

Table 16.11: Table Representation of Targets Array

	1	2	3...	...10	11	12
1997	192284420	86376721	260386468	125401565	72551855	136042505
1998	108645734	147656369	202158055	275721406	230488351	24901752
1999	85730444	74168740	39955768	141528519	178043261	181668256

PROC PRINT output shows the new data set *Work.Lookupl*, which contains the actual cargo values **`(RevCargo)`** and the target cargo values **`(Ctarget).`**

Work.Lookupl (first ten observations)

```
proc print data=work.lookup1 (obs=10);
   var salemon revcargo ctarget;
run;
```

Obs	SalePilon	RevCargo	Ctarget

1	JAN1997	$171,520,869.10	$192,284,420.00
2	JAN1998	$238,786,807.60	$108,645,734.00
3	JAN1999	$280,350,393.00	$85,730,444.00
4	FEB1997	$177,671,530.40	$86,376,721.00
5	FEB1998	$215,959,695.50	$147,656,369.00
6	FEB1999	$253,999,924.00	$74,168,740.00
7	MAR1997	$196,591,375.20	$28,526,103.00
8	MAR199S	$239,056,025.55	$202,158,055.00
9	MAR1999	$281,433,310.00	$39,955,768.00
10	APR1997	$380,304,120.20	$260,386,468.00

Using PROC TRANSPOSE

Overview

In the previous section, we compared actual revenue values to target revenue values using an array as a lookup table. Remember that

- *Sasuser.Monthsum* has an observation for each month and year.

Table 16.12: SAS Data Set Sasuser.Monthsum (first five observations of selected variables)

SaleMon	RevCargo	MonthNo
JAN1997	$171,520,869.10	1
JAN1998	$238,786,807.60	1
JAN1999	$280,350,393.00	1
FEB1997	$177,671,530.40	2
FEB1998	$215,959,695.50	2

- *Sasuser.Ctargets* has one variable for each month and one observation for each year.

Table 16.13: SAS Data Set Sasuser.Ctargets (selected variables)

Year	Jan	Feb	Mar	Apr	May	Jun
1997	192284420	86376721	28526103	260386468	109975326	102833104
1998	108645734	147656369	202158055	41160707	264294440	267135485
1999	85730444	74168740	39955768	312654811	318149340	187270927

Using arrays was a good solution because the orientation of the data sets differed. An alternate solution is to transpose *Sasuser.Ctargets* using PROC TRANSPOSE, and then merge the transposed data set with *Sasuser.Monthsum* by the values of **`Year`** and **`Month`.**

General form, PROC TRANSPOSE:

```
PROC TRANSPOSE <DATA=input-data-set>
        <(OUT=output-data-set>
        <NAME=variable-name>
        <PREFIX=variable-name>;
     BY <DESCENDING> variable-1
        <...<DESCENDING>variable-n>
        <NOTSORTED>;
     VAR variable(s);
```

```
RUN;
```

where

DATA=*input-data-set*

names the SAS data set to transpose.

OUT=output-data-set

names the output data set.

NAME=variable-name

specifies the name for the variable in the output data set that contains the name of the variable that is being transposed to create the current observation.

PREFIX=variable-name

specifies a prefix to use in constructing names for transposed variables in the output data set. For example, if PREFIX= *VAR*, the names of the variables are VAR1, VAR2, …,VAR*n*

BY statement

is used to transpose each BY group.

VAR *variable(s)*

names one or more variables to transpose.

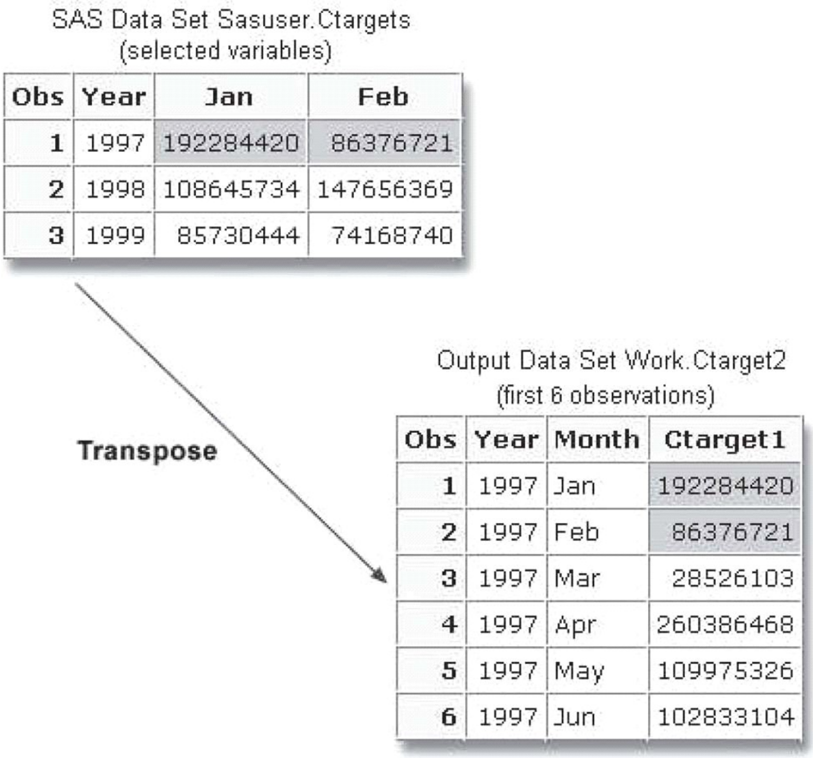
Note If *output-data-set* does not exist, PROC TRANSPOSE creates it by using the DATA *n* naming convention.

Note If you omit the *VAR* statement, the TRANSPOSE procedure transposes all of the numeric variables in the input data set that are not listed in another statement.

Note You must list character variables in a *VAR* statement if you want to transpose them.

The TRANSPOSE procedure creates an output data set by restructuring the values in a SAS data set. When the data set is restructured, selected variables are transposed into observations. The TRANSPOSE procedure can often eliminate the need to write a lengthy DATA step to achieve the same result. The output data set can be used in subsequent DATA or PROC steps for analysis, reporting, or further data manipulation.

PROC TRANSPOSE does not print the output data set. Use PROC PRINT, PROC REPORT, or some other SAS reporting tool to print the output data set.



SAS Data Set Sasuser.Ctargets
(selected variables)

Obs	Year	Jan	Feb
1	1997	192284420	86376721
2	1998	108645734	147656369
3	1999	85730444	74168740

Output Data Set Work.Ctarget2
(first 6 observations)

Obs	Year	Month	Ctarget1
1	1997	Jan	192284420
2	1997	Feb	86376721
3	1997	Mar	28526103
4	1997	Apr	260386468
5	1997	May	109975326
6	1997	Jun	102833104

Example

The following program transposes the SAS data set *Sasuser.Ctargets*. The OUT= option specifies the name of the output data set, *Work.Ctarget2*. All of the variables in *Sasuser.Ctargets* are transposed because all of the variables are numeric and a VAR statement is not used in the program.

```
proc transpose data=sasuser.ctargets
     out=work.ctarget2;
run;
```

Table 16.14: Input Data Set Sasuser.Ctargets (selected variables)

Year	Jan	Feb	Mar	Apr	May	Jun
1997	192284420	86376721	28526103	260386468	109975326	102833104
1998	108645734	147656369	202158055	41160707	264294440	267135485
1999	85730444	74168740	39955768	312654811	318149340	187270927

Table 16.15: Output Data Set: Work.Ctarget2

Obs	_NAME_	COL1	COL2	COL3
1	Year	1997	1998	1999
2	Jan	192284420	108645734	85730444
3	Feb	86376721	147656369	74168740
4	Mar	28526103	202158055	39955768
5	Apr	260386468	41160707	31265481
6	May	109975326	264294440	318149340
7	Jun	102833104	267135485	187270927
8	Jul	196728648	208694865	123394421
9	Aug	236996122	83456868	34273985
10	Sep	112413744	286846554	151565752
11	Oct	125401565	275721406	141528519

12	Nov	72551855	230488351	178043261
13	Dec	136042505	24901752	181668256

Notice that in the output data set, the variables are named **_NAME_, COL1, COL2**, and **COL3.**

NAME is the default name of the variable that PROC TRANSPOSE creates to identify the source of the values in each observation in the output data set. This variable is a character variable whose values are the names of the variables that are transposed from the input data set. For example, in *Work.Ctarget2* the values in the first observation in the output data set come from the values of the variable **Year** in the input data set.

The remaining transposed variables are named **COL1**...**COLn** by default. In *Work.Ctarget2*, the values of the variables **COL1, COL2**, and **COL3** represent the target cargo revenue for each month in the years 1997, 1998, and 1999.

Adding Descriptive Variable Names

You can use PROC TRANSPOSE options to give the variables in the output data set more descriptive names. The *NAME= option* specifies a name for the **_NAME_** variable.

The *PREFIX= option* specifies a prefix to use in constructing names for transposed variables in the output data set. For example, if PREFIX=*Ctarget*, the names of the variables are **Ctarget1, Ctarget2**, and **Ctarget3.**

```
proc transpose data=sasuser.ctargets
     out=work.ctarget2
     name=Month
     prefix=Ctarget;
run;
```

Table 16.16: Output Data Set: Work.Ctarget2

Obs	Month	Ctarget1	Ctarget2	Ctarget3
1	Year	1997	1998	1999
2	Jan	192284420	108645734	85730444
3	Feb	86376721	147656369	74168740
4	Mar	28526103	202158055	39955768
5	Apr	260386468	41160707	31265481
6	May	109975326	264294440	318149340
7	Jun	102833104	267135485	187270927
8	Jul	196728648	208694865	123394421
9	Aug	236996122	83456868	34273985
10	Sep	112413744	286846554	151565752
11	Oct	125401565	275721406	141528519
12	Nov	72551855	230488351	178043261
13	Dec	136042505	24901752	181668256

Note The RENAME=data set option can also be used with PROC TRANSPOSE to change variable names.

```
proc transpose data=sasuser.ctargets
     out=work.ctarget2 (rename=(col1=Ctarget1
         col2=Ctarget2 col3=Ctarget3))
     name=Month;
run;
```

The default label for the **_NAME_** variable is **NAME OF FORMER VARIABLE.** To see this, print the transposed data set using PROC PRINT with the LABEL option. You can use a LABEL statement to override the default label.

```
proc transpose data=sasuser.ctargets
     out=work.ctarget2
     name=Month
     prefix=Ctarget;
run;
```

```
proc print data=work.ctarget2 label;
label Month=MONTH;
run;
```

Merging the Transposed Data Set

Structuring the Data for a Merge

Remember that the transposed data set, *Work.Ctarget2*, needs to be merged with *Sasuser.Monthsum* by the values of **`Year`** and **`Month.`** Neither data set is currently structured correctly for the merge.

Table 16.17: SAS Data Set: Work.Ctarget2 (first five observations)

Obs	Month	Ctarget1	Ctarget2	Ctarget3
1	Year	1997	1998	1999
2	Jan	192284420	108645734	85730444
3	Feb	86376721	147656369	74168740
4	Mar	28526103	202158055	39955768
5	Apr	260386468	41160707	31265481

Table 16.18: SAS Data Set Sasuer.Monthsum (first five observations of selected variables)

Obs	SaleMon	RevCargo	MonthNo
1	JAN1997	$171,520,869.10	1
2	JAN1998	$238,786,807.60	1
3	JAN1999	$280,350,393.00	1
4	FEB1997	$177,671,530.40	2
5	FEB1998	$215,959,695.50	2

Using a BY Statement with PROC TRANSPOSE

In order to correctly structure *Work.Ctarget2* for the merge, a BY statement needs to be used with PROC TRANSPOSE. For each BY group, PROC TRANSPOSE creates one observation for each variable that it transposes. The BY variable itself is not transposed.

The following program transposes *Sasuser.Ctargets* by the value of **`Year.`** The resulting output data set, *Work.Ctarget2*, now contains 12 observations for each each year (1997, 1998, and 1999).

```
proc transpose data=sasuser.ctargets
     out=work.ctarget2
     name=Month
     prefix=Ctarget;
     by year;
run;
```

Table 16.19: Input Data Set Sasuser.Ctargets (selected variables)

Obs	Year	Jan	Feb	Mar	Apr	May	Jun
1	1997	192284420	86376721	28526103	260386468	109975326	102833104
2	1998	108645734	147656369	202158055	41160707	264294440	267135485
3	1999	85730444	74168740	39955768	312654811	318149340	187270927

Table 16.20: Output Data Set Work.Ctarget2 (first 12 observations)

Obs	Year	Month	Ctarget1
1	1997	Jan	192284420
2	1997	Feb	86376721
3	1997	Mar	28526103
4	1997	Apr	260386468
5	1997	May	109975326
6	1997	Jun	102833104
7	1997	Jul	196728648
8	1997	Aug	236996122
9	1997	Sep	112413744
10	1997	Oct	125401565
11	1997	Nov	72551855
12	1997	Dec	136042505

Caution The original SAS data set must be sorted or indexed before using a BY statement with PROC TRANSPOSE unless you use the NOTSORTED option.

Sorting the Work.Ctarget2 Data Set

The last step in preparing *Work.Ctarget2* for the merge is to use the SORT procedure to sort the data set by **`Year`** and **`Month`** as shown in the following program:

```
proc sort data=work.ctarget2;
   by year month;
run;
```

Notice that in the sorted version of *Work.Ctarget2*, the values of month are sorted alphabetically by year.

Table 16.21: SAS Data Set Work.Ctarget2 (sorted, first 12 observations)

Obs	Year	Month	Ctarget1
1	1997	Apr	260386468
2	1997	Aug	236996122
3	1997	Dec	136042505
4	1997	Feb	86376721
5	1997	Jan	192284420
6	1997	Jul	196728648
7	1997	Jun	102833104
8	1997	Mar	28526103
9	1997	May	109975326
10	1997	Nov	72551855
11	1997	Oct	125401565
12	1997	Sep	112413744

Reorganizing the Sasuser.Monthsum Data Set

The data in *Sasuser.Monthsum* must also be reorganized for the merge because the month and year values in that data set are combined in the variable **`SaleMon.`**

Table 16.22: SAS Data Set Sasuser.Monthsum (first five observations of selected variables)

Obs	SaleMon	RevCargo	MonthNo
1	JAN1997	$171,520,869.10	1
2	JAN1998	$238,786,807.60	1
3	JAN1999	$280,350,393.00	1
4	FEB1997	$177,671,530.40	2
5	FEB1998	$215,959,695.50	2

The following program creates two new variables, **`Year`** and **`Month`**, to hold the year and month values. The values for **`Year`** are created from **`SaleMon`** using the INPUT and SUBSTR functions. The values for **`Month`** are extracted from **`SaleMon`** using the LOWCASE and SUBSTR functions.

```
data work.mnthsum2;
   set sasuser.monthsum(keep=SaleMon RevCargo);
   length Month $ 8;
   Year=input(substr(SaleMon,4),4.);
   Month=substr(SaleMon,1,1)
         ||lowcase(substr(SaleMon,2,2));
run;
```

Table 16.23: SAS Data Set Work.Mnthsum2 (first six observations)

Obs	SaleMon	RevCargo	Month	Year
1	JAN1997	$171,520,869.10	Jan	1997
2	JAN1998	$238,786,807.60	Jan	1998
3	JAN1999	$280,350,393.00	Jan	1999
4	FEB 1997	$177,671,530.40	Feb	1997
5	FEB 1998	$215,959,695.50	Feb	1998
6	FEB 1999	$253,999,924.00	Feb	1999

Sorting the Work.Mnthsum2 Data Set

As with *Work.Ctarget2*, the last step in preparing for the merge is to sort the data set by the values of **`Year`** and **`Month`** as shown in the following program:

```
proc sort data=work.mnthsum2;
   by year month;
run;
```

Notice that in the sorted version of *Work.Mnthsum2*, the values of month are sorted alphabetically by year.

Table 16.24: SAS Data Set Work.Mnthsum2 (sorted, first twelve observations)

Obs	SaleMon	RevCargo	Month	Year
1	APR1997	$380,804,120.20	Apr	1997
2	AUG1997	$196,639,501.10	Aug	1997
3	DEC1997	$196,504,413.00	Dec	1997
4	FEB 1997	$177,671,530.40	Feb	1997
5	JAN1997	$171,520,869.10	Jan	1997
6	JUL1997	$197,163,278.20	Jul	1997
7	JUN1997	$190,560,828.50	Jun	1997
8	MAR1997	$196,591,378.20	Mar	1997
9	MAY1997	$196,261,573.20	May	1997
10	NOV1997	$190,228,066.70	Nov	1997

11	OCT1997	$196,957,153.40	Oct	1997
12	SEP1997	$190,535,012.50	Sep	1997

Completing the Merge

When the data is structured correctly, *Work.Mnthsum2* and *Work.Ctarget2* can be merged by the values of **`Year`** and **`Month`** as shown in the following program:

```
data work.merged;
   merge work.mnthsum2 work.ctarget2;
   by year month;
run;
```

Table 16.25: SAS Data Set Work.Mnthsum2 (first five observations)

Obs	SaleMon	RevCargo	Month	Year
1	APR1997	$380,804,120.20	Apr	1997
2	AUG1997	$196,639,501.10	Aug	1997
3	DEC1997	$196,504,413.00	Dec	1997
4	FEB 1997	$177,671,530.40	Feb	1997
5	JAN1997	$171,520,869.10	Jan	1997

Table 16.26: SAS Data Set Work.Ctarget2 (first five observations)

Obs	Year	Month	Ctarget1
l	1997	Apr	260386468
2	1997	Aug	236996122
3	1997	Dec	136042505
4	1997	Feb	86376721
5	1997	Jan	192284420

PROC PRINT output shows the resulting data set *Work.Merged*. The values of **`RevCargo`** represent the actual cargo revenue for each month. The values of **`Ctarget1`** represent the target cargo values for each month.

```
proc print
     data=work.merged (obs=10);
   format ctargetl dollar15.2;
   var month year revcargo ctarget1;
run;
```

SAS Data set Work.Merged (partial output)				
Obs	Month	Year	RevCargo	Ctarget1
1	Apr	1997	$380,804,120.20	$260,386,468.00
2	Aug	1997	$196,639,501.10	$236.996.122.00
3	Dec	1997	$196,504,413.00	$136,042,505.00
4	Feb	1997	$177,671,530.40	$86,376,721.00
5	Jan	1997	$171,520,869.10	$192,284,420.00
6	Jul	1997	$197,163,278.20	$196,728,648.00
7	Jun	1997	$190,560,828.50	$102,833,104.00
8	Mar	1997	$196,591,378.20	$26,526,103.00
9	May	1997	$196,261,573.20	$109,975,326.00

10	Nov	1997	$190,228,066.70	$72,551,855.00

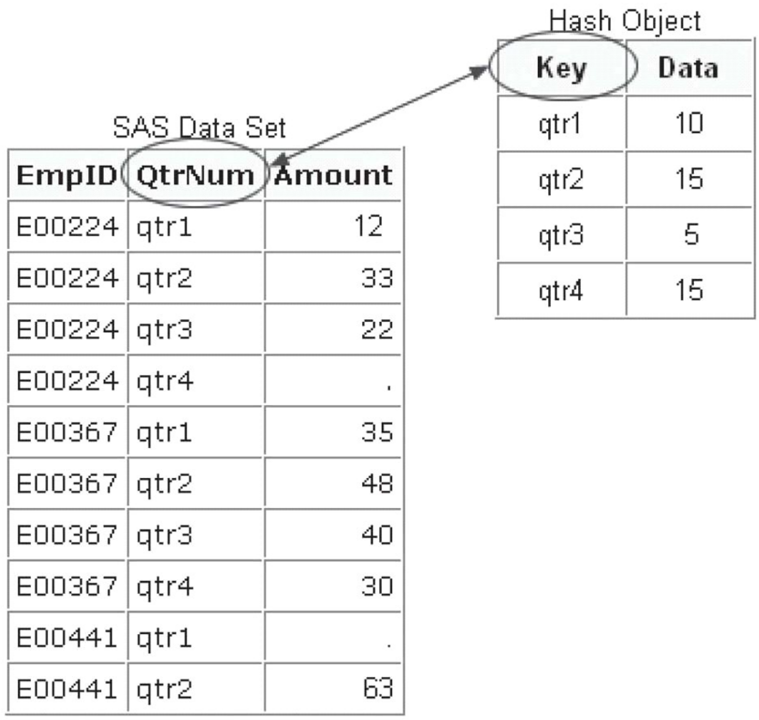
Using Hash Objects as Lookup Tables

Overview

Beginning with SAS 9, the hash object is available for use in a DATA step. The hash object provides an efficient, convenient mechanism for quick data storage and retrieval.

Unlike an array, which uses a series of consecutive integers to address array elements, a hash object can use any combination of numeric and character values as addresses. A hash object can be loaded from hard-coded values or a SAS data set, is sized dynamically, and exists for the duration of the DATA step.

The hash object is a good choice for lookups using unordered data that can fit into memory because it provides in-memory storage and retrieval and does not require the data to be sorted or indexed.



The Structure of a Hash Object

When a lookup operation depends on one or more key values, you can use the hash object. A hash object resembles a table with rows and columns and contains a key component and a data component.

The key component

- might consist of numeric and character values
- maps key values to data rows
- must be unique
- can be a composite.

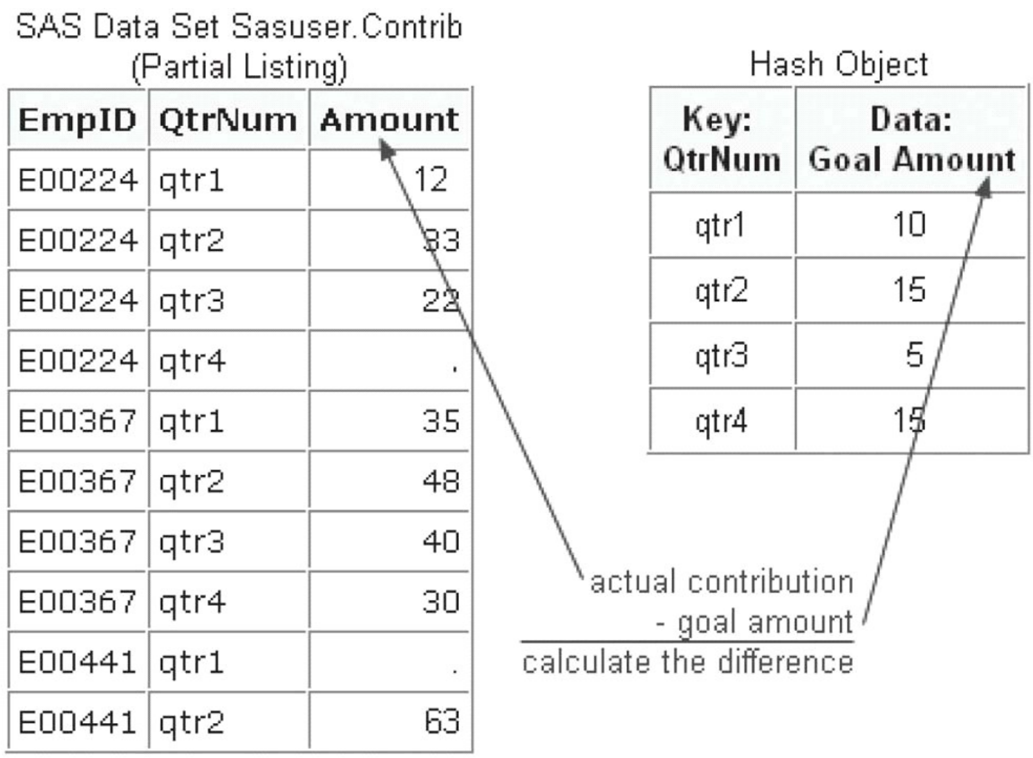
The data component

- can contain multiple data values per key value

- can consist of numeric and character values.

Example

Suppose you have a data set, named *Sasuser.Contrib*, that lists the quarterly contributions to a retirement fund. You can use the hash object to calculate the difference between the actual contribution and the goal amount.



The following program creates a hash object that stores the quarterly goal for employee contributions to the retirement fund. To calculate the difference between actual contribution and the goal amount, the program retrieves the goal amount from the hash object based on the key values.

```
data work.difference (drop= goalamount);
   length goalamount 8;
   if _N_ = 1 then do;
      declare hash goal( );
      goal.definekey("QtrNum");
      goal.definedata("GoalAmount");
      goal.definedone( );
      call missing(qtrnum, goalamount);
      goal.add(key:'qtr1', data:10 );
      goal.add(key:'qtr2', data:15 );
      goal.add(key:'qtr3', data: 5 );
      goal.add(key:'qtr4', data:15 );
   end;
   set sasuser.contrib;
   goal.find();
   Diff = amount - goalamount;
run;
```

We will see how the hash object is set up.

DATA Step Component Objects

The hash object is a DATA step component object. Component objects are data elements that consist of attributes and methods. *Attributes* are the properties that specify the information that is associated with an object. An example is size. *Methods* define the operations that an object can perform.

To use a DATA step component object in your SAS program, you must first declare and create (instantiate) the object.

Declaring the Hash Object

You declare a hash object using the DECLARE statement.

General form, DECLARE statement:

DECLARE *object variable <(<argument_tag-1: value-1<, …argument_tag-n: value-n>>)>;*

object

specifies the component object.

variable

specifies the variable name for the component object.

arg_tag

specifies the information that is used to create an instance of the component object.

value

specifies the value for an argument tag. Valid values depend on the component object,

where

Valid values for *object* are as follows:

- *hash* indicates a hash object.
- *hiter* indicates a hash iterator object.

Note The hash iterator object retrieves data from the hash object in ascending or descending key order.

The following DECLARE statement creates a hash object named *Goal*.

```
data work.difference (drop= goalamount);
   length goalamount 8;
   if _N_ = 1 then do;
      declare hash goal;
```

At this point, you have declared the variable *Goal*. It has the potential to hold a component object of the type hash.

Note The DECLARE statement is an executable statement.

Instantiating the Hash Object

You use the _NEW_ statement to instantiate the hash object.

General form, _NEW_ statement:

variable =**_NEW_***object(<argument_tag-l: value-l<, …argumentjag-n: value-n>>);*

where

variable

specifies the variable name for the component object.

object

specifies the component object.

argument_tag

specifies the information that is used to create an instance of the component object.

value

specifies the value for an argument tag. Valid values depend on the component object.

Valid values for *object* are as follows:

- *hash* indicates a hash object.
- *hiter* indicates a hash iterator object.

The following _NEW_ statement creates an instance of the hash object and assigns it to the variable *Goal*.

```
data work.difference (drop= goalamount);
   length goalamount 8;
   if _N_ = 1 then do;
      declare hash goal;
      goal= _new_ hash();
```

Declaring and Instantiating the Hash Object in a Single Step

As an alternative to the two-step process of using the DECLARE and _NEW_ statements to declare and instantiate a component object, you can use the DECLARE statement to declare and instantiate the component object in one step.

```
data Work.Difference (drop= goalamount);
   length goalamount 8;
   if _N_ = 1 then do;
      declare hash Goal();
```

Defining Keys and Data

Remember that the hash object uses lookup keys to store and retrieve data. The keys and the data are DATA step variables that you use to initialize the hash object by using dot notation method calls.

General form, dot notation method calls:

object.method(<argument_tag-l: value-l<, …argumentjag-n: value-n>>);

where

object

specifies the variable name for the DATA step component object.

method

specifies the name of the method to invoke.

argument-tag

identifies the arguments that are passed to the method.

value

specifies the argument value.

A key is defined by passing the key variable name to the DEFINEKEY method. Data is defined by passing the data variable name to the DEFINEDATA method. When all key and data variables are defined, the DEFINEDONE method is called. Keys and data can consist of any number of character or numeric DATA step variables.

The following code initializes the key variable **`QtrNum`** and the data variable **`GoalAmount.`**

```
data work.difference (drop= goalamount);
```

```
   length goalamount 8;
   if _N_ = 1 then do;
      declare hash goal();
      goal.definekey ("QtrNum");
      goal.definedata ("GoalAmount");
      goal.definedone();
```

Using the Call Missing Routine

The hash object does not assign values to key variables, and the SAS compiler cannot detect the implicit key and data variable assignments done by the hash object. Therefore, if no explicit assignment to a key or data variable appears in the program, SAS issues a note stating that the variables are uninitialized.

To avoid receiving these notes, use the CALL MISSING routine with the key and data variables as parameters. The CALL MISSING routine assigns a missing value to the specified character or numeric variables.

```
data Work.Difference (drop= goalamount);
length GoalAmount 8;
if _N_ = 1 then do; declare hash goal();
   goal.definekey("QtrNum");
   goal.definedata("GoalAmount");
   goal.definedone();
   call missing(qtrnum, goalamount);
```

Note Another way to avoid receiving notes stating that the variables are uninitialized is to provide an initial assignment statement that assigns a missing value to each key and data variable.

Loading Key and Data Values

So far, you have declared and instantiated the hash object, and initialized the hash object's key and data variables. You are now ready to store data in the hash object using the ADD method. The following code uses the ADD method to load the key values *qtr1, qtr2, qtr3*, and *qtr4* and the corresponding data values *10, 15, 5*, and *15* into the hash object.

```
data work.difference (drop= goalamount);
length goalamount 8;
if _N_ = 1 then do; declare hash goal();
   declare hash goal( );
   goal.definekey("QtrNum");
   goal.definedata("GoalAmount");
   goal.definedone( );
   call missing(qtrnum, goalamount);
   goal.add(key:'qtr1', data:10 );
   goal.add(key:'qtr2', data:15 );
   goal.add(key:'qtr3', data: 5 );
   goal.add(key:'qtr4', data:15 );
end;
```

Retrieving Matching Data

You use the FIND method to retrieve matching data from the hash object. The FIND method returns a value that indicates whether the key is in the hash object. If the key is in the hash object, then the FIND method also sets the data variable to the value of the data item so that it is available for use after the method call.

```
data work.difference (drop= goalamount);
   length goalamount 8;
   if _N_ = 1 then do;
      declare hash goal( );
      goal.definekey("QtrNum");
      goal.definedata("GoalAmount");
      goal.definedone( );
      call missing(qtrnum, goalamount);
      goal.add(key:'qtr1', data:10 );
      goal.add(key:'qtr2', data:15 );
      goal.add(key:'qtr3', data: 5 );
      goal.add(key:'qtr4', data:15 );
   end;
   set sasuser.contrib;
   goal.find();
```

```
   Diff = amount - goalamount;
run;
```

Hash Object Processing

We will consider what happens when the program is submitted for execution.

```
data Work.Difference (drop= goalamount);
   length goalamount 8 QtrNum $8.;
   if _N_ = 1 then do;
      declare hash goal( );
      goal.definekey("QtrNum");
      goal.definedata("GoalAmount");
      goal.definedone( );
      call missing(qtrnum, goalamount);
      goal.add(key:'qtr1', data:10 );
      goal.add(key:'qtr2', data:15 );
      goal.add(key:'qtr3', data: 5 );
      goal.add(key:'qtr4', data:15 );
   end;
   set sasuser.contrib;
   goal.find();
   Diff = amount - goalamount;
run;
```

The program executes until the DATA step encounters the end of the line. PROC PRINT output shows the completed data set.

```
proc print data=work.difference;
run;
```

Obs	QtrNum	EmpID	Amount	Diff
1	qtr1	E00224	12	2
2	qtr2	E00224	33	18
3	qtr3	E00224	22	17
4	qtr4	E00224	-	-
5	qtr1	E00367	35	25
6	qtr2	E00367	48	33
7	qtr3	E00367	40	35
8	qtr4	E00367	30	15
9	qtr1	E00441	-	-
10	qtr2	E00441	63	48

Creating a Hash Object from a SAS Data Set

Suppose you need to create a report that shows revenue, expenses, profits, and airport information. You have two data sets that contain portions of the required data. The SAS data set *Sasuser.Revenue* contains flight revenue information. The SAS data set *Sasuser.Acities* contains airport data including the airport code, location, and name.

Table 16.27: SAS Data Set Sasuser.Revenue (first five observations)

Origin	Dest	FlightID	Date	Rev1st	RevBusiness	RevEcon
ANC	RDU	IA03400	02DEC1999	15829	28420	68688
ANC	RDU	IA03400	14DEC1999	20146	26460	72981
ANC	RDU	IA03400	26DEC1999	20146	23520	59625
ANC	RDU	IA03401	09DEC1999	15829	22540	58671
ANC	RDU	IA03401	21DEC1999	20146	22540	65826

Table 16.28: SAS Data Set Sasuser.Acities (first five observations)

City	Code	Name	Country
Auckland	AKL	International	New Zealand
Amsterdam	AMS	Schiphol	Netherlands
Anchorage, AK	ANC	Anchorage International Airport	USA
Stockholm	ARN	Arlanda	Sweden
Athens (Athinai)	ATH	Hellinikon International Airport	Greece

To create the report, you can use a hash object to retrieve matching airport data from *Sasuser.Acities*.

Using a Non-Executing SET Statement

To initialize the attributes of hash variables that originate from an existing SAS data set, you can use a non-executing SET statement.

Because the IF condition is false during execution, the SET statement is compiled but not executed. The PDV is created with the variable **Code, City**, and **Name** from *Sasuser.Acities*.

```
data work.report;
   if _N_=1 then do;
      if 0 then
         set sasuser.acities (keep=Code City Name);
```

When you use this technique, the MISSING routine is not required.

Working with Multiple Data Variables

The hash object that you worked with earlier in this chapter contains one key variable and one data variable. In this example, you need to associate more than one data value with each key.

In the following code, the DECLARE statement creates the *Airports* hash object and loads it from *Sasuser.Acities*. the DEFINEKEY method call defines the key to be the value of the variable **Code.** The DEFINEDATA method call defines the data to be the values of the variables **City** and **Name.**

```
data work.report;
     if 0 then
        set sasuser.acities (keep=Code City Name);
    if _N_=1 then do;
    declare hash airports (dataset: "sasuser.acities")
    airports.definekey ("Code");
    airports.definedata ("City", "Name");
    airports.definedone();
end;
```

Table 16.29: HashObjectAirports

Key: Code	Data: City	Data: Name
ANC	Anchorage, AK	Anchorage International Airport
BNA	Nashville, TN	Nashville International Airport
CDG	Paris	Charles de Gaulle
LAX	Los Angeles, CA	Los Angeles International Airport
RDU	Raleigh-Durham, NC	Raleigh-Durham International Airport

Note To define all data set variables as data variables for the hash object, use the ALL: "YES" option.

hashobject.OEFWEDAlA (ALL:"YES");

Note The hash object can store multiple key variables as well as multiple data variables.

Retrieving Multiple Data Values

You can use multiple FIND method calls to retrieve multiple data values. In the following program, the FIND method calls retrieve the value of **`City`** and **`Name`** from the Airports hash object based on the value of **`Origin`**.

```
data work.report;
   if _N_=1 then do;
        if 0 then set sasuser.acities (keep=Code City Name);
      declare hash airports (dataset: "sasuser.acities");
      airports.definekey ("Code");
      airports.definedata ("City", "Name");
      airports.definedone();
   end;
 set sasuser.revenue;
 airports.find(key:origin);
 OriginCity=city;
 OriginAirport=name;
 airports.find(key:dest);
DestCity=city;
DestAirport=name;
run;
```

PROC PRINT output shows the completed data set.

```
proc print data=work.report;
   var origin dest flightid date origincity originairport
      destcity destairport;
run;
```

Obs	Origin	Dest	FlightID	Date	OriginCity	OriginAirport	DestCity	DestAirport
1	ANC	RDU	IA03400	02DEC1999	Anchcrage, AK	Anchorage International Airport	Raleigh-Durham, NC	Raleigh-Durham International Airport
2	ANC	RDU	IA03400	14DEC1999	Anchorage, AK	Anchorage International Airport	Raleigh-Durham, NC	Raleigh-Durham International Airport
3	ANC	RDU	IA03400	26DEC1999	Anchorage, AK	Anchorage International Airport	Raleigh-Durham, NC	Raleigh-Durham International Airport
4	ANC	RDU	IA03401	09DEC1999	Anchorage, AK	Anchorage International Airport	Raleigh-Durham, NC	Raleigh-Durham International Airport
5	ANC	RDU	IA03401	21DEC1999	Anchorage, AK	Anchorage International Airport	Durharn, NC	Raleigh-Durham International Airport

Figure 16.4: Partial Output (first five observations of selected variables)

Using Return Codes with the FIND Method

Method calls create a return code that is a numeric value. The value specifies whether the method succeeded or failed. A value of *0* indicates that the method succeeded. A non-zero value indicates that the method failed.

If the program does not contain a return code variable for the method call and the method fails, then an appropriate error message is written to the log.

To store the value of the return code in a variable, specify the variable name **`rc`** at the beginning of the method call. For example:

```
rc=hashobject.find (key:keyvalue);
```

The return code variable value can be used in conditional logic to ensure that the FIND method found a KEY value in the hash object that matches the KEY value from the PDV.

Example

Error messages are written to the log when the following program is submitted.

```
data work.report;
   if _N_=1 then do;
```

```
          if 0 then set sasuser.acities (keep=Code City Name);
      declare hash airports (dataset: "sasuser.acities");
      airports.definekey ("Code");
      airports.definedata ("City", "Name");
      airports.definedone();
   end;
set sasuser.revenue;
airports.find(key:origin);
OriginCity=city;
OriginAirport=name;
airports.find(key:dest);
DestCity=city;
DestAirport=name;
run;
```

Table 16.30: SAS Log

```
NOTE: There were 50 observations read from the data set SASUSER.ACITIES.
ERROR: Key not found.
ERROR: Key not found.
ERROR: Key not found.
ERROR: Key not found.
ERROR: Key not found.
ERROR: Key not found.
NOTE: The SAS System stopped processing this step because of errors.
NOTE: There were 142 observations read from the data set SASUSER.REVENUE.
WARNING: The data set WORK.REPORT1 may be incomplete. When this step was
         stopped there were 142 observations and 14 variables.
```

A closer examination of the output shows that the data set *Work.Report* contains errors. For example, notice that in observations 6 through 8 the value of both **OriginCity** and **DestCity** is *Canberra, Australian C* and the values of **OriginAirport** and **DestAirport** are missing.

The errors occur because the *Airports* hash object does not include the key value *WLG* or a corresponding **Name** value for the key value *CBR*.

Obs	Origin	Dest	FlightID	Date	OriginCity	OriginAirport	DestCity	DestAirport
6	CBR	WLG	IA1O5O0	04DEC1999	Canberra, Australian C		Canberra, Australian C	
7	CBR	WLG	IA10500	16DEC1999	Canberra, Australian C		Canberra, Australian C	
8	CBR	WLG	IA10500	2BDEC1999	Canberra, Australian C		Canberra, Australian C	

Figure 16.5: SAS Data Set Work.Report (observations 6 through 8 of selected variables)

Conditional logic can be added to the program to create blank values if the values loaded from the input data set, *Sasuser.Revenue*, cannot be found in the *Airports* hash object:

- If the return code for the FIND method call has a value of *0*, indicating that the method succeeded, the value of **City** and **Name** are assigned to the appropriate variables **(OriginCity** and **OriginAirport** or **DestCity** and **DestAirport).**
- If the return code for the FIND method call has a non-zero value, indicating the method failed, the value of **City** and **Name** are assigned blank values.

```
data work.report;
   if _N_=1 then do;
          if 0 then set sasuser.acities(keep=Code City Name);
      declare hash airports (dataset: "sasuser.acities");
      airports.definekey ("Code");
      airports.definedata ("City", "Name");
      airports.definedone();
end;
```

```
set sasuser.revenue;
rc=airports.find(key:origin);
if rc=0 then do;
    OriginCity=city;
    OriginAirport=name;
end;
else do;
   OriginCity='';
   OriginAirport='';
end;
rc=airports.find(key:dest);
if rc=0 then do;
   DestCity=city;
   DestAirport=name;
end;
else do;
   DestCity='';
   DestAirport='';
   end;
run;
```

PROC PRINT output shows the completed data set. Notice that in observations 6 through 8, the value of **`DestCity`** is now blank and no error messages appear in the log.

```
proc print data=work.report;
   var origin dest flightid date origincity originairport
      destcity destairport;
run;
```

Obs	Origin	Dest	FlightID	Date	OriginCity	OriginAirport	DestCity	DestAirport
1	ANC	RDU	IA03400	02DEC1999	Anchorage, AK	Anchorage International Airport	Raleigh-Durham, NC	Raleigh-Durham International Airport
2	ANC	RDU	IA03400	14DEC1999	Anchorage, AK	Anchorage International Airport	Raleigh-Durham, NC	Raleigh-Durham International Airport
3	ANC	RDU	IA03400	26DEC1999	Anchcrage. AK	Anchorage International Airport	Raleigh-Durham, NC	Raleigh-Durham International Airport
4	ANC	RDU	IA03401	09DEC1999	Anchcrage, AK	Anchorage International Airport	Raleigh-Durham, NC	Raleigh-Durham International Airport
5	ANC	RDU	IA03401	21DEC1999	Anchcrage, AK	Anchorage International Airport	Raleigh-Durham, NC	Raleigh-Durham International Airport
6	CBR	WLG	IA10500	04DEC1999	Canberra, Australian C			
7	CBR	WLG	IA10500	16DEC1999	Canberra, Australian C			
8	CBR	WLG	IA10500	28DEC1999	Canberra, Australian C			

Figure 16.6: SAS Data Set Work.Report (first eight observations of selected variables)

Table 16.31: SAS Log

```
NOTE: There were 50 observations read from the data set SASUSER.ACITIES.
NOTE: There were 142 observations read from the data set SASUSER.REVENUE.
NOTE: The data set WORK.REPORT2 has 142 observations and 15 variables.
```

Summary

Contents

This section contains the following topics.

Text Summary

Introduction

Sometimes, you need to combine data from two or more sets into a single observation in a new data set according to the values of a common variable. When data sources do not have a common structure, you can use a lookup table to match them.

Using Multidimensional Arrays

When a lookup operation depends on more than one numerical factor, you can use a multidimensional array. You use an ARRAY statement to create an array. The ARRAY statement defines a set of elements that you plan to process as a group.

Using Stored Array Values

In some cases, you might need to store array values in a SAS data set rather than loading them in an ARRAY statement. Array values should be stored in a SAS data set when

- there are too many values to initialize easily in the array
- the values change frequently
- the same values are used in many programs.

The first step in using stored array values is to create an array to hold the values in the target data set. The next step is to load the array elements. One method for accomplishing this task is to load the array within a DO loop. The last step is to read the actual values stored in the other source data set.

Using PROC TRANSPOSE

The TRANSPOSE procedure can also be used to match data when the orientation of the data sets differs. PROC TRANSPOSE creates an output data set by restructuring the values in a SAS data set, thereby transposing selected variables into observations. The transposed (output) data set can then be merged with the other data set in order to match the data.

The output data set contains several default variables.

- `_NAME_` is the default name of the variable that PROC TRANSPOSE creates to identify the source of the values in each observation in the output data set. This variable is a character variable whose values are the names of the variables that are transposed from the input data set. To override the default name, use the NAME= option.
- The remaining transposed variables are named **`COL1`**…**`COL`***n* by default. To override the default names, use the PREFIX= option.

Merging the Transposed Data Set

You might need to use a BY statement with PROC TRANSPOSE in order to correctly structure the data for a merge. For each BY group, PROC TRANSPOSE creates one observation for each variable that it transposes. The BY variable itself is not transposed. In order to structure the data for a merge, you might also need to sort the output data set. Any other source data sets might need to be reorganized and sorted as well. When the data is structured correctly, the data sets can be merged.

Using Hash Objects as Lookup Tables

Beginning with SAS 9, the hash object is available for use in a DATA step. The hash object provides an efficient, convenient mechanism for quick data storage and retrieval.

A hash object resembles a table with rows and columns; it contains a key component and a data component. Unlike an array, which uses a series of consecutive integers to address array elements, a hash object can use any combination of numeric and character values as addresses.

The hash object is a DATA step component option. Component objects are data elements that consist of attributes and methods. To use a DATA step component object in your SAS program, you must first declare and create (instantiate) the object. After you define the hash object's key and data variables, you can load the variables from hard-coded values or a SAS data set.

You use the FIND method call return code that is a numeric value. The value specifies whether the method succeeded or failed. A value of 0 indicates that the method succeeded. A nonzero value indicates that the method failed. The return code variable value can be used in conditional logic to ensure that the FIND method found KEY value in the hash object that matches the KEY value from the PDV.

Syntax

Using a Multidimensional Array

```
LIBNAME libref 'SAS-data-library';
DATA librefsas-data-set(DROP=variable(s));
ARRAY array-name {rows,cols,…} <$><length>
       <array-elements> <{initial values)>;
    SET<SAS-data-set(s)<(data-set-options(s))>><options>;
    variable=expression;
    variable=array-name {rows,cols,…};
RUN;
```

Using Stored Array Values

```
LIBNAME libref'SAS-data-library';
DATA libref.sas-data-set;
ARRAY array-name 1 {rows,cols,…} <$><length>
        <array-elements> <(initial values)>;
    IF expression THEN DO index-variable=specification;
       SET <SAS-data-set(s)<{data-set-options(s) )>> <options>;
       ARRAY array-name2 {rows,cols,…} <$> <length>
              <array-elements> <(initial values)>;
       DO index-variable=specification;
             array-namel {rows,cols,…}=array-name2 {rows,cols,…};
       END;
    END;
RUN;
```

Using PROC TRANSPOSE and a Merge

```
LIBNAME libref'SAS-data-library';
PROC TRANSPOSE <DATA=input-data-sef>
       <OUT=output-data-sef>
       <NAME=name>
       <PREFIX=prefix>;
     BY <DESCENDING> variable-1
          <…<DESCENDING>variable-n><NOTSORTED>;
RUN;
PROC SORT;
     BY <DESCENDING> variable-1
        <…<DESCENDING>variable-n>;
RUN;
DATA libref.sas-data-set;
     SET<SAS-data-set(s)<(data-set-option(s))>><options>;
     LENGTH variable(s) <$> length;
     variable=expression;
     variable=expression;
RUN;
PROC SORT;
```

```
     BY <DESCENDING> variable-1 <...<DESCENDING>variable-n>;
RUN;
DATA libref.sas-data-set;
     MERGE SAS-data-set-1 <(data-set-options)>
              SAS-data-set-2<(data-set-options)>;
     BY <DESCENDING> variable-1 <...<DESCENDING>variable-n#x003E;;
RUN;
```

Using a Hash Object as a Lookup Table

```
DATA libref.sas-data-set;
     IF expression THEN statement;
        DECLARE object variable <(<argument_tag-l: value-1<, …argument_tag-n: value-n>>)>;
        variable = _NEW_object(<argument_tag-l: value-1<, …argument_tag-n: value-n>>);
        object.DEFINEKEY('keyvarname-1'<…, 'keyvarname-n'>);
        object.DEFINEDATA('datavarname-l'<,… 'datavarname-n'>);
        object.DEFINEDONE();
        CALL MISSING(varname1<, varname2, …>);
        object.ADD(<KEY:keyvalue-l,…,KEY:keyvalue-n,DATA:datavalue-1, …DATA:datavalue-n>
END;
SET <SAS-data-set(s)<(data-set-options(s) )>><options>;
object.FIND(<KEY:keyvalue-1,…,KEY:keyvalue-n>);
RUN;
```

Sample Programs

Using a Multidimensional Array

```
data work.wndchill (drop = column row);
  array WC{4,2} _temporary_
     (-22,-16,-28,-22,-32,-26,-35,-29);
  set sasuser.flights;
  row = round(wspeed,5)/5;
  column = (round(temp,5)/5)+3;
  WindChill= wc{row,column};
run;
```

Using Stored Array Values

```
data work.lookup1;
   array Targets{1997:1999,12} _temporary_;
   if _n_=1 then do i= 1 to 3;
      set sasuser.ctargets;
      array Mnth{*} Jan--Dec;
      do j=1 to dim(mnth);
         targets{year,j}=mnth{j};
      end;
   end;
   set sasuser.monthsum(keep=salemon revcargo monthno);
   year=input(substr(salemon,4),4.);
   Ctarget=targets{year,monthno};
   format ctarget dollar15.2;
run;
```

Using PROC TRANSPOSE and a Merge

```
proc transpose data=sasuser.ctargets
     out=work.ctarget2
     name=Month
     prefix=Ctarget;
   by year;
run;

proc sort data=work.ctarget2;
   by year month;
run;

data work.mnthsum2;
   set sasuser.monthsum(keep=SaleMon RevCargo);
   length Month $ 8;
   Year=input(substr(SaleMon,4),4.);
```

```
   Month=substr(SaleMon,1,1)
         ||lowcase(substr(SaleMon,2,2));
run;

proc sort data=work.mnthsum2;
   by year month;
run;
data work.merged;
   merge work.mnthsum2 work.ctarget2;
   by year month;
run;
```

Using a Hash Object That Is Loaded From Hard-Coded Values

```
data work.difference (drop= goalamount);
   length goalamount 8;
   if _N_ = 1 then do;
      declare hash goal( );
      goal.definekey("QtrNum");
      goal.definedata("GoalAmount");
      goal.definedone( );
      call missing(qtrnum, goalamount);
      goal.add(key:'qtr1', data:10 );
      goal.add(key:'qtr2', data:15 );
      goal.add(key:'qtr3', data: 5 );
      goal.add(key:'qtr4', data:15 );
   end;
   set sasuser.contrib;
   goal.find();
   Diff = amount - goalamount;
run;
```

Using a Hash Object That Is Loaded From a SAS Data Set

```
data work.report;
   if _N_=1 then do;
           if 0 then set sasuser.acities(keep=Code City Name);
     declare hash airports (dataset: "sasuser.acities");
     airports.definekey ("Code");
     airports.definedata ("City", "Name");
     airports.definedone();
end;
set sasuser.revenue;
rc=airports.find(key:origin);
if rc=0 then do;
   OriginCity=city;
   OriginAirport=name;
end;
else do;
   OriginCity='';
   OriginAirport='';
end;
rc=airports.find(key:dest);
if rc=0 then do;
   DestCity=city;
   DestAirport=name;
end;
else do;
   DestCity='';
   DestAirport='';
end;
run;
```

Points to Remember

- The name of an array must be a SAS name that is not the name of a SAS variable in the same DATA step.
- Array elements must be either all numeric or all character.
- The initial values specified for an array can be numbers or character strings. You must enclose all character strings in

quotation marks.

- The original SAS data set must be sorted or indexed prior to using a BY statement with PROC TRANSPOSE unless you use the NOTSORTED option.
- The hash object is a good choice for lookups using unordered data that can fit into memory because it provides in-memory storage and retrieval and does not require the data to be sorted.
- The hash object is sized dynamically, and exists for the duration of the DATA step.

Quiz

Select the best answer for each question. After completing the quiz, check your answers using the answer key in the appendix.

1. Which SAS statement correctly specifies the array **`Sales`** as illustrated in the following table? ?

Table Representation of Sales Array

m1	m2	m3	m4
m5	m6	m7	m8
m9	m10	m11	m12

a. `array Sales{3,4} m1-m12;`

b. `array Sales{4,3} m1-m12;`

c. `array {3,4} Sales m1-m12;`

d. `array {4,12} Sales m1-m12;`

2. Which of the following statements creates temporary array elements? ?

a. `array new {*} _temporary_;`

b. `array new {6} _temporary_;`

c. `array new {*} _temporary_ Jan Feb Mar Apr May Jun;`

d. `array _temporary_ new {6} Jan Feb Mar Apr May Jun;`

3. Which DO statement processes all of the elements in the **`Yearx`** array? ?

```
array Yearx{12} Jan--Dec;
```

a. `do i=1 to dim(yearx);`

b. `do i=1 to 12;`

c. `do i=Jan to Dec;`

d. *a* and *b*

4. Given the following program, what is the value of **`Points`** in the fifth observation in the data set *Work.Results?* ?

SAS Data Set Work.Contest

```
data work.results;
   array score{2,4} _temporary_
         (40,50,60,70,40,50,60,70);
   set work.contest;
   Points=score{week,finish};
run;
```

Obs	Name	Week	Finish
1	Tuttle	1	1

2	Gomez	1	2
3	Chapman	1	3
4	Venter	1	4
5	Vandeusen	2	1
6	Tuttle	2	2
7	Venter	2	3
8	Gomez	2	4

a. *40*

b. *50*

c. *60*

d. *70*

5. Array values should be stored in a SAS data set when ?

a. there are too many values to initialize easily in an array.

b. the values change frequently.

c. the same values are used in many programs.

d. all of the above

6. Given the following program, which statement is not true? ?

```
data work.lookup1;
   array Targets{1997:1999,12} _temporary_;
   if _n_=1 then do i= 1 to 3;
               set sasuser.ctargets;
               array Mnth{*} Jan--Dec;
               do j=1 to dim(mnth);
                  targets{year,j}=mnth{j};
               end;
   end;
   set sasuser.monthsum(keep=salemon revcargo monthno);
   year=input(substr(salemon,4),4.);
   Ctarget=targets{year,monthno};
run;
```

a. The IF-THEN statement specifies that the **`Targets`** array is loaded once.

b. During the first iteration of the DATA step, the outer DO loop executes three times.

c. After the first iteration of the DO loop, the pointer drops down to the second SET statement.

d. During the second iteration of the DATA step, the condition **`_N_=1 is false.`** So, the DO loop does not execute.

7. Given the following program, which variable names will appear in the data set *Work.New?* ?

SAS Data Set Work.Revenue

```
proc transpose
     data=work.revenue
     out=work.new;
run;
```

Obs	Year	Jan	Feb	Mar	Apr
1	2000	1.003.561	922.080	836.068	973.016
2	2001	1.078.552	1.013.798	1.047.812	1.005.575
3	2002	1.182.442	1.657.323	1.079.866	1.466.640

a. `Year, Jan, Feb, Mar, Apr`

b. `Year, 2000, 2001, 2002`

c. `_NAME_, Col1, Col2, Col3`

d. `_NAME_, Jan, Feb, Mar, Apr`

8. Which program creates the output data set *Work.Temp2?* ?

SAS Data Set Work.Temp

Obs	Month1	Month2	Month3
1	13604250	24901752	18166825
2	72551855	23048835	17804326
3	12540156	27572140	14152851
4	11241374	28684655	15156575

SAS Data Set Work.Temp2

Obs	Month	Quarter1	Quarter2	Quarter3	Quarter4
1	Month1	13604250	72551855	12540156	11241274
2	Month2	24901752	23048835	27572140	28684655
3	Month3	18166825	17804326	14152851	15156575

a.
```
a. proc transpose data=work.temp
        out=work.temp2
        prefix=Quarter;
   run;
```

b.
```
b. proc transpose data=work.temp
        out=work.temp2
        name=Month
        prefix=Quarter;
   run;
```

c.
```
c. proc transpose data=work.temp
        out=work.temp2
        prefix=Month
        name=Quarter;
   run
```

d.
```
d. proc transpose data=work.temp
        out=work.temp2
        prefix=Month
        index=Quarter;
   run;
```

9. Which version of the data set *Work.Sales2* is created by the following program? ?

SAS Data Set Work.Sales

```
proc transpose data=work.sales
     out=work.sales2
     name=Week;
     by employee;
run;
```

Obs	Employee	Weekl	Week2
1	Almers	3393.50	2192.25
2	Bona venture	5093.75	2247.50
3	Johnson	1813.30	2082.75
4	LaMance	1572.50	2960.00

Obs	Week	COL1	COL2	COL3	COL4

a.

1	Week1	3393.50	5093.75	1813.30	1572.5
2	Week2	2192.25	2247.50	2028.75	2960.0

b.

Obs	Employee	Week	COL1
1	Almers	Week1	3393.50
2	Almers	Week2	2192.25
3	Bonnaventure	Week1	5093.75
4	Bonnaventure	Week2	2247.50
5	Johnson	Week1	1813.30
6	Johnson	Week2	2028.75
7	LaMance	Week1	1572.50
8	LaMance	Week2	2960.00

c.

Obs	Week	Almers	Bonnaventure	Johnson	LaMance
1	Week1	3393.50	5093.75	1813.30	1572.5
2	Week2	2192.25	2247.50	2028.75	2960.0

d.

Obs	Employee	_NAME_	Week
1	Almers	Week1	3393.50
2	Bonnaventure	Week1	5093.75
3	Johnson	Week1	1813.30
4	LaMance	Week1	1572.50
5	Almers	Week2	2192.25
6	Bonnaventure	Week2	2247.50
7	Johnson	Week2	2028.75
8	LaMance	Week2	2960.00

10. Which program creates the data set *Work.Fishsize?* ?

SAS Data Set Work.Fishdata

Obs	Location	Date	Length1	Weight1	Length2	Weight2
1	Cole Pond	02JUN95	31	0.25	32	0.30
2	Cole Pond	04AUG95	29	0.23	30	0.25
3	Eagle Lake	02JUN95	32	0.35	32	0.25
4	Eagle Lake	04AUG95	33	0.30	33	0.28

SAS Data Set Work.Fishsize

Obs	Location	Date	_NAME_	Measurement1
1	Cole Pond	02JUN95	Length 1	31.00
2	Cole Pond	02JUN95	Weightl	0.25
3	Cole Pond	02JUN95	Length 2	32.00
4	Cole Pond	02JUN95	Weight2	0.30
5	Cole Pond	04AUG95	Length 1	29.00
6	Cole Pond	04AUG95	Weight1	0.23
7	Cole Pond	04AUG95	Length 2	30.00
8	Cole Pond	04AUG95	Weight2	0.25
9	Eagle Lake	02JUN95	Length 1	32.00

10	Eagle Lake	02JUN95	Weight1	0.35
11	Eagle Lake	02JUN95	Length 2	32.00
12	Eagle Lake	02JUN95	Weight2	0.25
13	Eagle Lake	04AUG95	Length 1	33.00
14	Eagle Lake	04AUG95	Weight1	0.30
15	Eagle Lake	04AUG95	Length 2	33.00
16	Eagle Lake	04AUG95	Weight2	0.28

a.
```
a. proc transpose data=work.fishdata
        out=work.fishsize
        prefix=Measurement;
   run;
```

b.
```
b. proc transpose data=work.fishdata
        out=work.fishsize
        prefix=Measurement;
        by location;
   run;
```

c.
```
c. proc transpose data=work.fishdata
        out=work.fishsize
        prefix=Measurement;
        by date;
   run;
```

d.
```
d. proc transpose data=work.fishdata
        out=work.fishsize
        prefix=Measurement;
        by location date;
   run;
```

Answers

1. Correct answer: a

   An array is specified using the keyword ARRAY followed by the name of the array and the dimensions of the array. In a two-dimensional array, the two dimensions can be thought of as a table of rows and columns. The first dimension in the ARRAY statement specifies the number of rows. The second dimension specifies the number of columns.

2. Correct answer: b

   To create temporary array elements, specify the keyword TEMPORARY_ after the array name and dimension. Remember that if you use an asterisk to count the array elements, you must list the array elements. You cannot use the asterisk and the _TEMPORARY_ keyword together in an ARRAY statement.

3. Correct answer: d

   To process all of the elements in an array, you can use either the DIM function with the array name as the argument or specify the array dimension.

4. Correct answer: a

   The ARRAY statement creates the two-dimensional array **`score`** and specifies the dimensions of the array: two rows and four columns. The value of **`Points`** for each observation is determined by referencing the array based on the values of **`Week`** and **`Finish`** in the *Work.Contest* data set. The row number for the array reference is determined by the value of **`Week`**. The column number for the array reference is determined by the value of **`Finish`**.

5. Correct answer: d

   Array values should be stored in a SAS data set when there are too many values to initialize easily in an array, the values change frequently, or the same values are used in many programs.

6. Correct answer: c

   The IF-THEN statement specifies that the **`Targets`** array is loaded only once, during the first iteration of the DATA step. During the first iteration of the DATA step, the condition **`_N_=1`** is true, so the outer DO loop executes three times; once for each observation in *Sasuser.Ctargets*. After the third iteration of the DO loop, the pointer drops down to the second SET statement and the values from the first observation in Sasuser.Monthum are read into the program data vector. During the second iteration of the DATA step, the condition _**`N`**_=1 is false. So, the DO loop doesn't execute.

7. Correct answer: c

   The TRANSPOSE procedure creates an output data set by restructuring the values in a SAS data set. When the data set is restructured, selected variables are transposed into observations. The procedure creates several variable names by default. _**`NAME`**_ is the default name of the variable that PROC TRANSPOSE creates to identify the source of the values in each observation in the output data set. The remaining transposed variables are named **`COL1`**…**`COLn`** by default.

8. Correct answer: b

   You can use several options with PROC TRANSPOSE to give the variables in the output data set descriptive names. The NAME= option specifies a name for _**`NAME`**_ variable. The PREFIX= option specifies a prefix to use in constructing names for the other variables in the output data set.

9. Correct answer: b

   A BY statement can be used with PROC TRANSPOSE. For each BY group, PROC TRANSPOSE creates one observation for each variable that it transposes. The BY variable itself is not transposed. The original data set must be sorted or indexed prior to using a BY statement with PROC TRANSPOSE.

10. Correct answer: d

    The observations in `Work.Fishsize` are grouped by `Location` and **`Date`**. For each BY group, PROC TRANSPOSE creates four observations, one for each variable (**`Length1, Weight1, Length2`**, and **`Weight2`**) that it is transposing.